
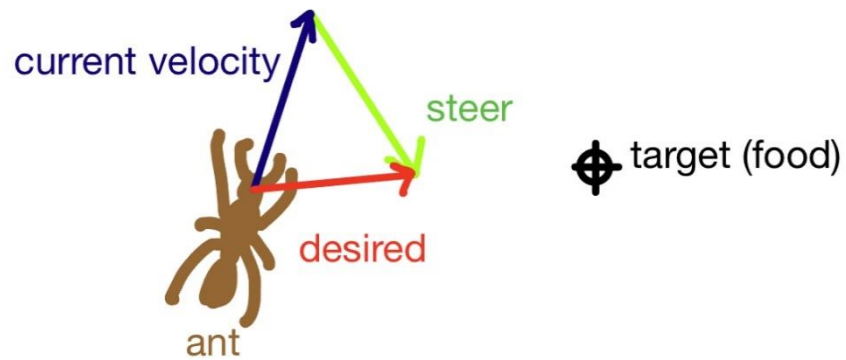


Algorithms explained using structured English

Below includes explanations of the other main complex algorithms in my system using English.

Function / procedure description	Code and notes
Ant Vision Circle	<p>The way ants find food is by using a “vision circle” which surrounds the ant. If food falls within this circle, then ant will acknowledge food and act. This is shown on the image on the left.</p> 
Ants sees food.	<p>This is when the ant wander mode is set to 1.</p> <p>Ant’s vision circle has collided with food on screen. This should call function <code>ant.set_mode(1)</code> which will change the ant’s wander mode to 1, the mode when the ant has seen food. Will also call function <code>ant.wander(food_bit.get_position())</code> which gets the foods position and returns it into the ant wander function. Inside the ant wander function, when its mode is set to 1, it will change the Ant’s “target” to coordinates of the food. The Ant’s target is what the point that the Ant attempts to move towards.</p> <pre>def seek(self): # Move towards target, movement looks natural self.__desired = (self.__target - self.__position).normalize() * self.__max_speed steer = (self.__desired - self.__velocity) if steer.length() > self.__seek_force: steer.scale_to_length(self.__seek_force)</pre> <p>Ant will then attempt to get to food item using code above.</p>



For this movement to look natural I have implemented it in a way as shown through my code. The Image above can be used to describe this. The “target” represents the food and when the ant sees this food, it will attempt to change its blue current velocity vector to the red “desired” arrow velocity vector. To make this natural, I have added a “steer” vector which is a vector that is between the current velocity and the desired velocity. This is controlled using the `self.__seek_force` attribute which is a fixed number that is used to control how quickly the ant changes to the desired velocity. This means that when an ant can sense food but is not directly looking at it, instead of changing its direction immediately, it slowly starts to face the food. When the ant is be

Ants wandering
behaviour

This is when the ant wander mode is set to 0.

```
self.__velocity = vector(self.__max_speed, 0).rotate(uniform(0, 360))
```

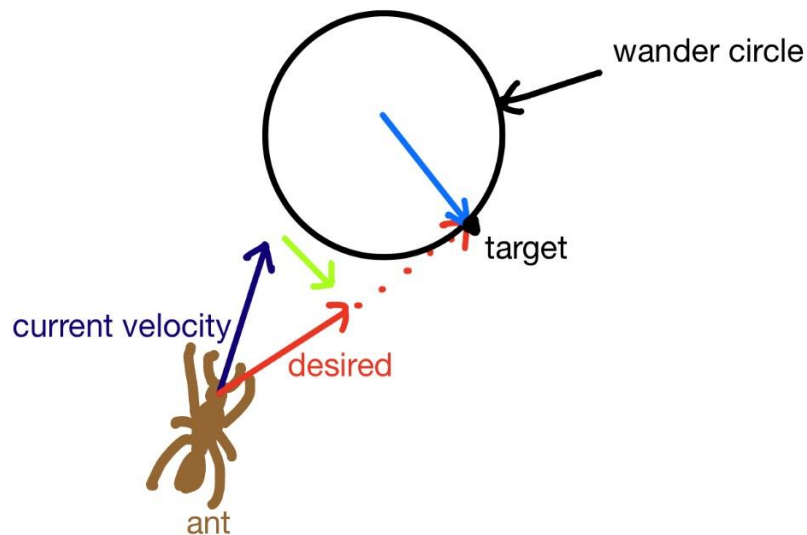
Ants have a velocity vector which determines how many pixels ant will move every frame and in a random direction. The attribute `self.__max_speed` is used to set a maximum speed the ant can move by per frame.

```
self.__acceleration = vector(0, 0)
```

Ants also have acceleration vector which represents how much velocity changes each frame.

```
self.__target = vector(randint(0, self.__wn_width), randint(0, self.__wn_height))
```

Ants also have a “target” which is set to a random coordinate within the screen. This target is used for the ant’s random movement.



The ant has an invisible “wander circle” in front of it and on this invisible circle there is a target point that the ant follows. This target point changes constantly, and the circle moves around randomly. The desired arrow will point towards this target point and then using the steer function from before will change the velocity. This creates a realistic randomly movement effect for the ant as it follows the randomly moving circle and constantly tries to look at different points on the circle.

In case the Ant happens to accidentally wander off the screen, the following code is used:

```
if (self.__position[0] > self.__wn_width) or (self.__position[0] < 0)
    or (self.__position[1] > self.__wn_height)
    or (self.__position[1] < 0):

    self.__target = vector(randint(0, self.__wn_width), randint(0,
self.__wn_height))
    return self.seek()
```

This randomises the position of the target again, so it is within the screen.

Ant Goes Home

This is when wander mode is set to 2.

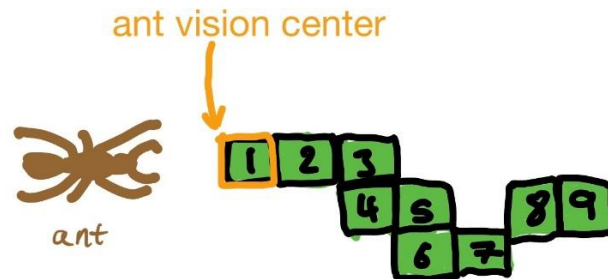
```
if self.__wander_mode == 2:
    self.__holding_food = True
    new_circle_position = self.__home + self.__velocity.normalize()
    * 200
```

	<pre> self.__target = new_circle_position + vector(60, 10).rotate(uniform(0, 360)) self.__acceleration = self.seek() self.__velocity += self.__acceleration if self.__velocity.length() > self.__max_speed: self.__velocity.scale_to_length(self.__max_speed) # If velocity exceeds max speed then cap the speed self.__position += self.__velocity </pre> <p>Once the ant has collected food and is ready to go home it will change its target attribute to the Home. To make this movement look natural, I have simply decided to change the position of the invisible circle that the ant follows to be on the home and the target will constantly change to different parts on the home so then it moves around whilst walking home instead of just in a straight line.</p>
<p>Pheromone Update</p>	<p>This is a function which is in the Pheromone class. There is a while loop which runs in the main script which calls this function for every ant.</p> <pre> def pheromone_update(self): if self.__fixed_pheromone and self.__ant.get_mode() == 2: self.__pheromone_strength = (1 / self.__ant.distance_from_home()) self.__fixed_pheromone = False if self.__ant.get_mode() == 2: self.ant_collision_with_cell(self.__ant.get_vision_center()) pixel_counter = -1 for i in self.__pheromoneStack: pixel_counter += 1 if pygame.time.get_ticks() - self.__pheromoneStack[pixel_counter][3] >= 5000: self.__pheromoneStack[pixel_counter][2] -= (1 - self.__evaporation_rate) * 1/self.__pheromone_strength/500 </pre> <p>If the ant's mode is set to 2 this means the Ant is going home and the program should add to the pheromone list. The strength of the pheromone is determined using a formula which considers the distance the ant is from home. Then <code>self.ant_collision_with_cell(self.__ant.get_vision_center())</code> is used to call a function that adds information to the pheromone list. Also goes through every item in pheromone list and if pheromone has been</p>

	<p>there for more than 5 seconds it starts to evaporate pheromone at a rate using formula.</p>
Adding Pheromone to grid.	<pre>def pheromone_add(self, row): self.__grid_matrix[row[0], row[1]] = row[2] if self.__grid_matrix[row[0], row[1]] > 254: self.__grid_matrix[row[0], row[1]] = 255 if self.__grid_matrix[row[0], row[1]] < 10: self.__grid_matrix[row[0], row[1]] = 0 pygame.gfxdraw.box(self.__screen, ((self.__tileSize * int(row[1])), (self.__tileSize * int(row[0])), self.__tileSize, self.__tileSize), (144, 238, 144, self.__grid_matrix[row[0], row[1]])) return self.__grid_matrix[row[0], row[1]]</pre> <p>This is a function in the PheromoneGrid class that takes in a list “row”. This contains information such as the Grid coordinates of pheromone, the pheromone strength at that coordinate and the time pheromone was drawn onto screen. The actual grid value only holds the strength of the pheromone <code>self.__grid_matrix[row[0], row[1]] = row[2]</code> # row[2] is the pheromone strength.</p> <p>These values are capped at 255 as the Pygame “gfxdraw” function allows a maximum opacity of 255. When pheromone is drawn, it is converted to its actual coordinate on the Pygame screen using: <code>((self.__tileSize * int(row[1])), (self.__tileSize * int(row[0])))</code> . This multiplies the grid coordinates by the tile size that is used in the Grid to convert it to its actual coordinates on the screen.</p>
Ant detecting pheromone	<pre>pheromone = Grid.collide_item(ant.get_vision_center()) if pheromone: follow_path = random() if follow_path > 0.05: ant.set_mode(3) ant.wander(pheromone)</pre> <p>The way Ants can detect pheromone to follow a pheromone path is done using the code above. I will soon go into more detail on the “collide_item” function, if a pheromone value is returned from the function the Ant will most likely follow the path. If it decides to follow the path, the ant’s wander mode will be set to 3. The wander mode of 3 works the same as the wander mode of 1 (when the ant sees food),</p>

Instead of the target being food, the target is now the pheromone they have detected.

This can be explained using the image below.



In this image, each green square represents a cell in the grid that has pheromone inside it. The Ant vision center is just a cell “selector” that is in front of the ant which the ant constantly checks to see if there is food on the cell that is selected. When this vision center collides with pheromone (in this case it has collided with the pheromone labelled 1), the Ant will make that individual cell of pheromone its target and follow it, it will then follow through each individual pheromone until it reaches 9. All the ant is doing is changing its target again and again when it has pheromone in front of it, this will eventually lead them to food and if not, then the ant will change back to wander mode 0 and start randomly walking around again. However, this comes with a problem as the ant vision center is just a cell on the grid so that means an ant can only see the cell that is directly in front of it. This means that once we reach pheromone number 3 on the image, it won't be able to see pheromone number 4 as it is towards the right instead of directly in front of it.

I have managed to find a fix to this problem using the code below in the PheromoneGrid class:

```
def collide_item(self, center):
    grid_center0 = int((self.__tileSize * round(center[0] /
self.__tileSize) / self.__tileSize))
    grid_center1 = int((self.__tileSize * round(center[1] /
self.__tileSize)) / self.__tileSize)
    sub_matrix = self.__grid_matrix[grid_center1 - 3:grid_center1 + 4,
grid_center0 - 3:grid_center0 + 4]
    value_dict = {}
```

```

for y_index, row in enumerate(sub_matrix):
    for x_index, value in enumerate(row):
        if value != 0 and value != -100:
            value_dict[(y_index, x_index)] = value

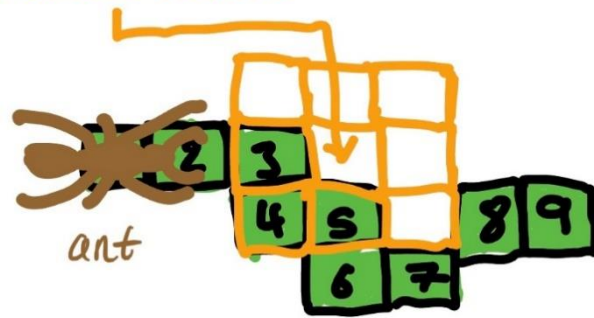
if len(value_dict) > 0:
    probabilities = list(value_dict.values())
    path = choices(list(value_dict.items()), weights=probabilities)
    y_index = path[0][0][0] - 3
    x_index = path[0][0][1] - 3
    grid_center1 += y_index
    grid_center0 += x_index
    return grid_center0 * self.__tileSize, grid_center1 *
self.__tileSize

```

What I have done to fix this problem is instead of giving the ant just a single cell to check for food, it has a whole mini 3x3 grid which is within the grid of the simulation. This 3x3 grid is created around the ant vision center selector using this line: `sub_matrix = self.__grid_matrix[grid_center1 - 3:grid_center1 + 4, grid_center0 - 3:grid_center0 + 4]`. After this, a nested for loop is done through these cells in the smaller grid to check if there is pheromone in any of the cells. The value 0 means there is nothing in the cell and the value -100 means there is a wall, so these are just ignored. If there is pheromone in the cell, then it is stored in a dictionary. Then, the ant chooses which cell to follow through a random choice but will most likely follow the cell that has a higher pheromone strength inside it (this is how an ant differentiates between which paths to take in real life) using this line: `path = choices(list(value_dict.items()), weights=probabilities)`. Once it has chosen the cell to follow, this value is converted to the actual coordinates on the screen and is returned so the ant makes this exact point its new target.

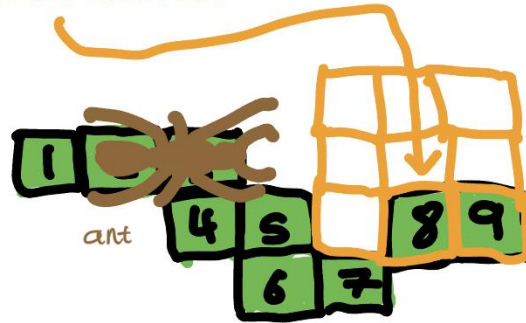
This can also be shown using the image below:

ant vision center



The ant chooses between cell 3, 4 and 5 to follow and if it happens to choose to follow 3, the grid will move forward by two spaces once it reaches 3. Then, it will choose between cell 8 and 9 as a target as these will be the two cells with pheromone inside them.

ant vision center



I have chosen to use a 3x3 grid for now, but I may choose to change this later to a bigger grid.

Ant Avoiding Wall

This is when ant wander mode is set to 5. Ants have a small wall detection circle in front of them as illustrated bellow (smaller circle).



```
def flee(self, target): # Code for ant avoiding wall
    steer = vector(0, 0)
    self.__desired = vector(0, 0)
    self.__dist = self.__position - target
    if self.__dist.length() < self.__flee_radius:
        self.__desired = self.__dist.normalize() * self.__max_speed
```



```
steer = self.__desired - self.__velocity
if steer.length() > self.__seek_force:
    steer.scale_to_length(self.__seek_force)
```

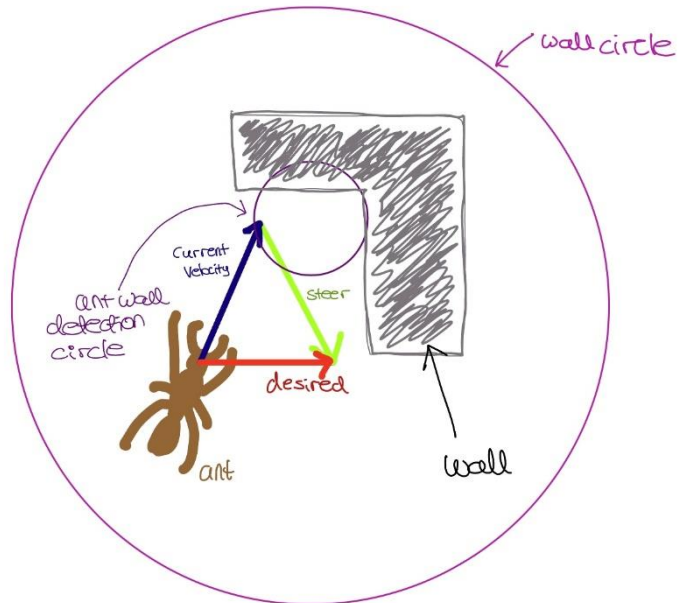
If this circle collides with the wall, then a signal is sent to the ant to change its wander mode to mode 5. Mode 5 will check if ant is within certain radius of a circle around the wall using `self.__dist = self.__position - target`. If this value falls within the radius, the ant will attempt to move away from this wall by doing the opposite of what it does when it sees food. The code is in fact similar to when the ant goes towards food except for the `self.__desired` vector which now goes in the opposite direction to make the ant avoid the wall instead. I have also (in other areas of the code) made sure to slow down the speed of the ant as it approaches a wall so that it does not accidentally walk over the wall as the max speed could be set too high, so the ant is so fast it just walks over the wall.

Once the ant has avoided the wall, it should be outside the wall radius, so this part of the code is run:

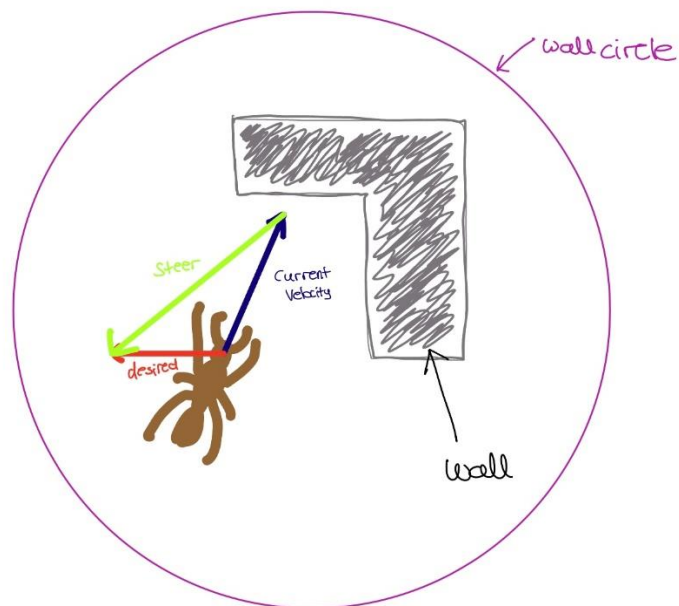
```
else:
    if self.__holding_food:
        self.set_mode(4)
        self.__initial_time = pygame.time.get_ticks()
    else:
        self.set_mode(0)
return steer
```

which will return the ant back to its previous state.

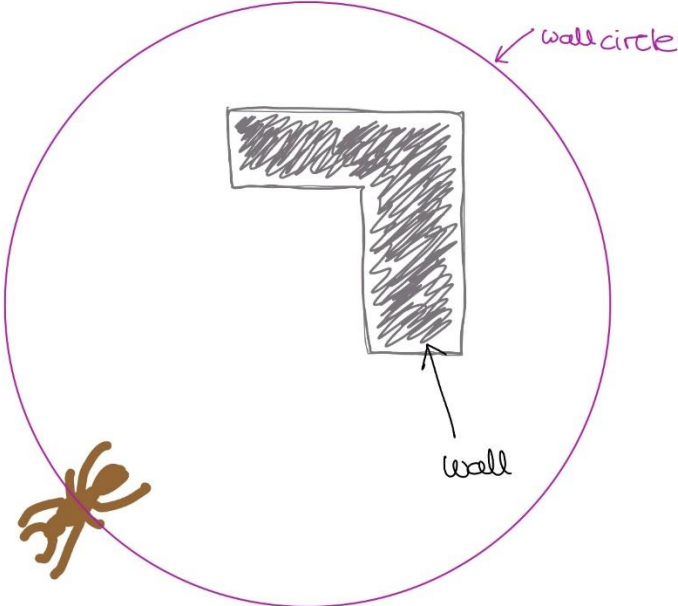
These steps can be shown using the below images.



1) Ant wall detection circle collides with wall and ant is within circle around wall.



2) Ant desired vector goes in the opposite direction so ant will start to steer away instead of go into wall.

	 <p>3) Ant Travels away and once it leaves the circle around the wall, it switches back to original state.</p> <hr/>
<p>Ant collides with wall whilst holding food</p>	<p>If an ant collides with a wall whilst holding food then the <code>self.__bounce</code> will increment by 1. When this increases it will change the angle at which the ant tries to get home through this line:</p> <pre>new_circle_position = self.__home + self.__velocity.normalize() * bounce</pre> <p>This basically adds a vector with length 1 which has a direction that will increase in size based on how many bounces there are and add this to homes coordinates.</p>